**Part 1: Algorithms**. Invent an algorithm named RankComponentsBySize that operates on a Partition object p (through its API) and produces a vector v of unsigned integers

such that v[i] is the size of the (1 +i)

th largest component of p: p[0] is the size of the largest

component, p[1] is the size of the second-largest component, and so on.

Also invent an algorithm that creates a Partition object p that captures the precise component structure of an undirected graph g. Combine the two algorithms to obtain an

application for a graph g: The Component Rank Sequence of g.

**The first part of the problem seems trivial, regardless of how many components we have. You can loop through the partition object p.components() times and use the given api functions to see how many elements are in each component and keep track of that. From there we can use a simple sorting algorithm to put them in order from largest to smallest.**

**The second part of where we need to create a partition object of an undirected graph can be implementedThe technology developed in Sections 3 and 4 shows that the following instantiation of the DFS algorithm produces a Vector<N> component such that component[x] is the component containing x for each vertex x of G. All that is needed is to declare the component vector and make a small post-processing adjustment to DFSurvey::Search():**

**void DFSurvey::Search()**

**{**

**unsigned components = 0;**

**for (each vertex v of g\_)**

**if (color[v] == white)**

**{**

**components +=1;**

**Search(v);**

**}**

**component[f] = components;**

**}**

**Recall that we know the DFS forest is a collection of trees, each tree generated by a call to Search(v). The DFS trees are in 1-1 correspondence to the components of G. The algorithm above counts the components and assigns each vertex its component number as it is processed.**

**This is an algorithm that runs in time Θ(|V| + |E|) and results in a mechanism for constant-time lookup of the component of any vertex.**